MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo No. 837

March, 1985
REVISED August, 1986

GPSG-Recognition is NP-Hard

Eric Sven Ristad

ABSTRACT:

Proponents of generalized phrase structure grammar (GPSG) cite its weak context-free generative power as proof of the computational tractability of GPSG-Recognition. Since context-free languages (CFLs) can be parsed in time proportional to the cube of the sentence length, and GPSGs only generate CFLs, it seems plausible that GPSGs can also be parsed in cubic time. This longstanding, widely-assumed GPSG "efficient parsability" result is misleading: parsing the sentences of an arbitrary GPSG is likely to be intractable, because a reduction from 3SAT proves that the universal recognition problem for the GPSGs of Gazdar (1981) is NP-hard. Crucially, the time to parse a sentence of a CFL can be the product of sentence length cubed and context-free grammar size squared, and the GPSG grammar can result in an exponentially large set of derived context-free rules. A central object in the 1981 GPSG theory, the metarule, inherently results in an intractable parsing problem, even when severely constrained. The implications for linguistics and natural language parsing are discussed.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research has been provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-80-C-0505. The author is indebted to Ed Barton, Robert Berwick, Noam Chomsky, James Higginbotham, Shaun Keller, Richard Larson, Alexis Manaster-Ramer, Stanley Peters, Geoff Pullum, Dave Waltz, Scott Weinstein, and an anonymous LI referee for assistance in writing this paper.

©Massachusetts Institute of Technology, 1985, 1986

1 Introduction

Proponents of generalized phrase structure grammar (GPSG) cite its weak context-free generative power as proof of the computational tractability of GPSG-Recognition. Since context-free languages (CFLs) can be parsed in time proportional to the cube of the sentence length, and GPSGs only generate CFLs, it seems plausible that GPSGs can also be parsed in cubic time. Gazdar (1981:155) argues that this would in turn provide "the beginnings of an explanation for the obvious, but largely ignored, fact that humans process the utterances they hear very rapidly."[1]

This widely-assumed GPSG "efficient parsability" result is misleading: parsing the sentences of an arbitrary GPSG is likely to be intractable, because a reduction from 3SAT proves that the universal recognition problem (RP) for the GPSGs of Gazdar (1981) is NP-hard. This complexity classification means that the fastest recognition algorithm for GPSGs could take at least exponential time. Therefore *nothing in the GPSG formal framework guarantees efficient parsability*, contrary to Gazdar's argument from weak context-free generative power. Crucially, the time to parse a sentence of a CFL can be the product of sentence length cubed and context-free grammar size squared, and the GPSG grammar can result in an exponentially large set of derived context-free rules. A central object in GPSG theory, the metarule, inherently results in an intractable parsing problem, even when severely constrained. Section 2.3 below contains a formal proof.

The apparent paradox between the efficiency of context-free parsing and the intractability of GPSG parsing is resolved below. I also discuss how the recognition problem is posed and the implications of this result for linguistics and natural language parsing.

2 Complexity of GPSG-Recognition

This section begins by formally specifying the class of generalized phrase structure grammars described by Gazdar (1981).[2] After providing some

[1] Joshi (1985:226) and Peterson (1985:315) make similar assumptions about the connection between weak context-free generative power and efficient processability.

[2] GPSG theory has changed considerably since Gazdar (1981). Gazdar, Klein, Pullum and Sag (1985), henceforth GKPS, contains a detailed and precise formal exposition of current GPSG theory. Ristad (1986a) analyzes the computational complexity of that

relevant technical background, I prove that the problem of determining for an arbitrary GPSG G and input string w whether w is in the language $L(G)$ generated by G is as hard as any nondeterministic polynomial time computation (NP-hard), and hence likely to be intractable.

2.1 Formal Specification of GPSG

The GPSGs of Gazdar (1981) contain sets of nonterminal symbols V_N, terminal symbols V_T, basic rules R, and metarules M. *Basic rules* are the set of rules required by a context-free grammar for English not handling unbounded dependencies. They are interpreted as node admissibility conditions rather than conventional context-free rewrite rules. A example basic rule is $1a$, which is equivalent to the rewrite rule $1b$.

$$\begin{array}{ll} a. & [_S\ NP\ VP\ ] \\ b. & [S \rightarrow NP\ VP\ ] \end{array} \tag{1}$$

Metarules are a grammar for generating a grammar. They typically express many of the linguistic relationships expressed by transformations in transformational grammar. Unlike transformations, whose domain can be an entire tree, the domain of metarules is effectively restricted to trees of depth one. Formally, they are functions from basic rules to sets of context-free rules with fixed input and output patterns that contain variables and constants. If a context-free rule matches the input pattern under some specialization of the metarule's variables, then the metarule generates a context-free rule corresponding to the metarule's output pattern under the same specialization of the metarule's variables. Metarule 2 performs Subject-Auxiliary inversion in Gazdar (1981):

$$\left[\begin{array}{c} \text{VP} \\ \left[\begin{array}{c} +\text{fin} \\ +\text{aux} \end{array} \right] \end{array} \rightarrow V\ X \right] \Longrightarrow \left[Q \rightarrow \begin{array}{c} \text{V} \\ \left[\begin{array}{c} +\text{fin} \\ +\text{aux} \end{array} \right] \end{array} NP\ X \right] \tag{2}$$

Metarule 2 states that for every rule which expands a finite VP and introduces a tensed auxiliary verb (e.g. the rule that generates structure 3), there

theory and proves that the universal RP for the GPSGs of GKPS can take more than exponential time, that is, time proportional to $c^{f(n)}$ for some constant c, polynomial $f(n)$, and input string and grammar size n. Section 3 below explores the practical implications of exponential time results.

is also a rule which expands the sentential category Q as that tensed auxiliary verb, followed by a NP and whatever else followed the tensed auxiliary in the original rule (e.g. the rule that generates 4).

$$[\underset{[+\text{fin}\ +\text{aux}]}{VP}\ [\underset{[+\text{fin}\ +\text{aux}]}{V}\ \text{is}\ ][_{AP}\ \text{stupid}\ ]] \tag{3}$$

$$[_{Q}[\underset{[+\text{fin}\ +\text{aux}]}{V}\ \text{is}\ ][_{NP}\ \text{Kim}\ ][_{AP}\ \text{stupid}\ ]] \tag{4}$$

The set of derived rules is closed under metarule application. That is, the complete set of derived rules in a GPSG consists of the basic rules plus the maximal rule set that can be arrived at by repeatedly applying each metarule to each derived rule.

Unconstrained metarule application may generate infinite sets of rules and describe arbitrary languages, including recursively enumerable ones. To preserve both the weak context-free generative power of a GPSG grammar and the supposedly attendant computational benefits, some formal constraints on metarules have been proposed in the GPSG literature. One proposal is to constrain variables in the metarule pattern to be "abbreviatory variables," i.e. variables that can only stand for strings in a finite and extrinsically determined range. Formally, each metarule variable may only range over a finite subset of $(V_N \cup V_T)^*$. While this constraint can affect the extensional language of the grammar in linguistically unmotivated and arbitrary ways, I adopt a stronger version of this constraint for the purposes of examining its computational implications. In the proof below, all metarule variables either are constants (e.g. $0, 1$) or stand for a single symbol and can only have two possible values (e.g. x can only stand for the symbols $0, 1$). See Shieber et.al. (1983) for a discussion of Gazdar's 1982 'abbreviatory variables' proposal and some other proposals for restricting metarules.

I further restrict the derivational power of metarules as follows. Metarules are functions from context-free rules to context-free rules (not from rules to sets of rules) that may only operate once in the derivation of a given rule. That is, a metarule pattern may match a rule in only one way, and metarules may not operate recursively on their own output. In addition, no unrecoverable deletion can occur in a derivation and no two metarules or basic rules may be identical either in pattern or function.

2.2 The Classifications of Complexity Theory

Mathematical complexity theory measures the intrinsic lower-bound difficulty of obtaining the solution to a problem no matter how the solution is obtained. Complexity theory studies the *structure of problems*: it classifies problems according to the amount of computational resources (e.g. time, space, electricity) needed to solve them on some abstract machine model (e.g. a deterministic Turing machine). Complexity classifications are invariant across a wide range of primitive machine models, all choices of representation, algorithm, and actual implementation, and even the resource measure itself. The robustness of these classifications is especially appropriate for cognitive science: while we do know what abstract problems the brain solves, we don't know much about the representations, algorithms, or hardware involved.

$\mathcal{P}$ is the natural and important class of problems solvable in deterministic $\mathcal{P}$olynomial time — problems with efficient solutions. $\mathcal{NP}$ is the class of all problems solvable in $\mathcal{N}$ondeterministic $\mathcal{P}$olynomial time. Informally, a problem is in $\mathcal{NP}$ if we can guess an answer to the problem and then verify its correctness in polynomial time. For example, the problem of deciding whether a whole number i is composite is in $\mathcal{NP}$ because it can be solved by guessing a pair of potential divisors, and then quickly checking if their product equals i. A problem T is *NP-hard* if it is at least as hard computationally as any problem in the class $\mathcal{NP}$: if we had a subroutine that solved T in polynomial time, then we could write a program to solve any problem in $\mathcal{NP}$ in polynomial time on a deterministic Turing machine. Note that T need not be in $\mathcal{NP}$ to be NP-hard. A problem is *NP-complete* if it is both in $\mathcal{NP}$ and NP-hard. NP-hard problems can be solved only by methods too slow for even the fastest computers. Since it is widely believed — though not proved — that no faster methods of solution can ever be found for these problems, NP-complete problems are considered computationally infeasible. A famous NP-complete problem is the traveling salesman problem, that is, to find the shortest route for a traveling salesman who must visit a number of cities and return to the city he started at. Lewis and Papadimitriou (1978) informally discuss these problems and other complexity issues. Barton, Berwick, and Ristad (1986, forthcoming) further explores the relationship between computational complexity and natural language.

Complexity classifications are established with the proof technique of reduction. A *reduction* converts instances of a problem T of known complexity

into instances of a problem S whose complexity we wish to determine. The reduction operates in polynomial time. Therefore, if we had a polynomial time algorithm for solving S, then we could also solve T in polynomial time, simply by converting instances of T into S. (This follows because the composition of two polynomial time functions is also polynomial time.) Formally, if we choose T to be NP-complete, then the polynomial time reduction shows that S is at least as hard as T, or NP-hard. If we were also to prove that S was in $\mathcal{NP}$, then S would be NP-complete.

In this case, the known NP-complete problem T is 3SAT, and the problem S of unknown complexity is GPSG-Recognition. Therefore, the proof will reduce instances of 3SAT (a 3-CNF Boolean formula F) into instances of GPSG-Recognition (a GPSG G and an input string x). The *3-Satisfiability problem* (3SAT) is to determine, given a boolean expression in 3-CNF, whether the formula is satisfiable. 3SAT is NP-complete. An example of a satisfiable 3-CNF boolean formula with five clauses is:

$$(a \vee b \vee c) \wedge (\overline{a} \vee d \vee e) \wedge (e \vee \overline{d} \vee \overline{c}) \wedge (b \vee c \vee d) \wedge (\overline{a} \vee \overline{d} \vee \overline{e})$$

A *Boolean expression* is an expression composed of variables (e.g. x), parentheses, and the logical operators $\vee$ (OR), $\wedge$ (AND), and negation. Negation is represented as a horizontal bar over the negated expression (e.g. $\overline{x}$ is the negation of the variable x). A *literal* is a variable or the negation of a variable. Variables may have the values 0 (false) and 1 (true), as do expressions. An expression is *satisfiable* if there is some assignment of 0's and 1's to the variables that gives the expression the value 1.

A Boolean expression is in *conjunctive normal form* (CNF) if it is of the form $E_1 \wedge E_2 \wedge \cdots \wedge E_k$ and each clause E_i is of the form $\alpha_{i1} \vee \alpha_{i2} \vee \cdots \vee \alpha_{im_i}$, where each α_{ij} is a *literal* — either a variable x or a negated variable $\overline{x}$. An expression is in *3-CNF* if each clause in the CNF expression contains exactly three distinct literals.

2.3 Reduction from 3SAT to GPSG-Recognition

Recall that a reduction is an algorithm for converting instances of one problem into instances of another problem. In the reduction below, it is important to distinguish the process by which the metarules are constructed (the reduction) from the process by which metarules are applied in GPSG to generate a set of context-free rules. In particular, the reduction does not

generate all possible truth assigments for the variables in F; this would of course require exponential time, and invalidate the reduction. The work of generating all possible truth assignments is done by the metarule application process, not by metarule construction. For example, in part 2.c of the following proof, the reduction tests w_i and $\overline{q_j}$ for equality in order to construct the metarules which instantiate negated variables. The reduction will require $O(m^3 \log m)$ time to construct the $|w|$ metarules. The constructed metarules, however, never test w_i and $\overline{q_j}$ for equality in the metarule application process.

Theorem 1 *GPSG-Recognition is NP-hard*

Proof. The proof will reduce 3-SAT to GPSG-Recognition in polynomial time. Assume as input a 3-CNF formula F of length m using the n variables $q_1, q_2, \ldots, q_n$. Let w be the string of formula literals in F; in general, w_i will denote the i^{th} symbol in the string w.

In the following reduction, S will be the distinguished start nonterminal of the constructed GPSG; 0 and 1 will be special *metarule constants*; a_i and b_j will be *metarule variables* that range over the metarule constants 0, 1; and $A, B, \#$, and ‡ will be special grammar symbols. I will use 0^i to denote the length i string of all 0's, where 0 is a metarule constant. Thus, 0^5 denotes 00000.

The reduction constructs a GPSG grammar G such that the special symbol # is an element of $L(G)$ iff F is satisfiable. The idea is that the constructed GPSG will guess an assignment of truth values for the formula variables, and then determine if the guessed assignment satisfies the formula F.

The constructed metarules will use the right-hand side of the context-free rules to record guesses and as a scratchpad during the evaluation of the guess. The string x to the left of the "‡" symbol represents the formula F, where x_i is the literal in the i_{th} position of debracketing of F, w. The string y to the right of the "‡" symbol encodes an assignment of truth values to variables, where the j^{th} position y_j stores the truth value assigned to the formula variable q_j.

To "guess" an assignment, the metarules will generate all possible truth assignments to the n variables, exactly as a deterministic Turing machine might if it were simulating a nondeterministic Turing machine. To verify

the guess, some metarules substitute the guessed variable values for the formula literals, and other metarules "evaluate" the instantiated formula. For the 3-CNF formula to be true, every clause must be true, and for a clause to be true, at least one of the three variables in the clause must be true. The following metarules exploit the regularity of 3-CNF formulas in a very obvious manner.

G contains

1. $n+1$ basic rules:

$$\text{the } i^{th} \text{ rule: } [A \rightarrow 0^{|w|} \ddagger 1^i 0^{n-i}] \quad \text{where } 0 \leq i \leq n$$

   The basic rules, in conjunction with some of the metarules, will generate all possible assignments to the formula variables.

2. the metarules

   (a) $\frac{1}{2}n(n-1)$ metarules, which generate all possible truth assignments when they are applied to the basic rules. (n is the number of distinct variables in the formula F.) For all i and for all j, $1 \leq i < j \leq n$, construct the metarule:

$$[A \rightarrow 0^{|w|} \ddagger a_1 \dots a_i \dots a_j \dots a_n] \implies$$
$$[A \rightarrow 0^{|w|} \ddagger a_1 \dots a_{i-1} a_j a_{i+1} \dots a_{j-1} a_i a_{j+1} \dots a_n]$$

   These metarules exchange any two symbols (in positions i and j) in a string of length n. Therefore, if one of these metarules applies to a rule, it will exchange the truth values assigned by the rule to the variables q_i and q_j. Since any subset of these metarules can apply one by one to the basic rules before the next metarule (immediately below) shuts off the process, any posssible truth-assignment to the variables can be encoded in the substring to the right of the "‡" symbol. (See the formal proof of lemma 1 below.)

   (b) One metarule, to stop the generation of truth assignments:

$$[A \rightarrow 0^{|w|} \ddagger a_1 \dots a_n] \implies [B \rightarrow 0^{|w|} \ddagger a_1 \dots a_n]$$

   This metarule prevents any of the metarules described above from changing a guess while the following metarules determine if the guess satisfies F.

(c) $|w|$ metarules are needed to instantiate the variable truth assignments generated by the preceeding metarules into the formula literals. Note that the basic rules assign formula literals the truth value 0 by default. Consequently, we only need to instantiate literals with the truth value 1. For each literal in F, there will be exactly one metarule: i will index the i^{th} literal in w, and j will index the corresponding formula variable, whose value is encoded in the string to the left of the ‡ symbol. A negative literal (e.g. $\overline{a}$) will be true when its variable is false, while a positive literal (e.g. a) will be true when its variable is true. Formally, include the following metarules for all i, $1 \leq i \leq |w|$:

- If $w_i = q_j$ for some j, then construct this metarule to watch for q_j being true:

$$[B \rightarrow a_1 \ldots a_{i-1} 0 a_{i+1} \ldots a_{|w|} \ddagger b_1 \ldots b_{j-1} 1 b_{j+1} \ldots b_n] \implies$$
$$[B \rightarrow a_1 \ldots a_{i-1} 1 a_{i+1} \ldots a_{|w|} \ddagger b_1 \ldots b_{j-1} 1 b_{j+1} \ldots b_n]$$

- Otherwise, $w_i = \overline{q_j}$ for some j, and we construct this metarule instead to watch for q_j being false:

$$[B \rightarrow a_1 \ldots a_{i-1} 0 a_{i+1} \ldots a_{|w|} \ddagger b_1 \ldots b_{j-1} 0 b_{j+1} \ldots b_n] \implies$$
$$[B \rightarrow a_1 \ldots a_{i-1} 1 a_{i+1} \ldots a_{|w|} \ddagger b_1 \ldots b_{j-1} 0 b_{j+1} \ldots b_n]$$

Note that these metarules instantiate the negation of formula variable q_j in the i^{th} position of w.

(d) Include $\frac{7}{3}|w|$ metarules to verify that the guessed assigment satisfies F. (Recall that the formula F is in 3-CNF, and therefore $|w|$ will be a multiple of 3.) According to the definition of 3SAT, a 3-CNF formula is true if all its clauses are true, and a clause is true if any of its three literals is true. Accordingly, these metarules will erase a 3-CNF clause iff at least one of its three literals is true. Therefore, if the metarules can erase the entire string to the left of the "‡" symbol, then the formula is satisfiable. There are seven such metarules for each k, $0 \leq k \leq \frac{|w|}{3} - 1$:

$$[B \rightarrow 001 a_1 \ldots a_{3k} \ddagger b_1 \ldots b_n] \implies [B \rightarrow a_1 \ldots a_{3k} \ddagger b_1 \ldots b_n]$$
$$[B \rightarrow 010 a_1 \ldots a_{3k} \ddagger b_1 \ldots b_n] \implies [B \rightarrow a_1 \ldots a_{3k} \ddagger b_1 \ldots b_n]$$
$$[B \rightarrow 100 a_1 \ldots a_{3k} \ddagger b_1 \ldots b_n] \implies [B \rightarrow a_1 \ldots a_{3k} \ddagger b_1 \ldots b_n]$$
$$[B \rightarrow 011 a_1 \ldots a_{3k} \ddagger b_1 \ldots b_n] \implies [B \rightarrow a_1 \ldots a_{3k} \ddagger b_1 \ldots b_n]$$
$$[B \rightarrow 101 a_1 \ldots a_{3k} \ddagger b_1 \ldots b_n] \implies [B \rightarrow a_1 \ldots a_{3k} \ddagger b_1 \ldots b_n]$$
$$[B \rightarrow 110 a_1 \ldots a_{3k} \ddagger b_1 \ldots b_n] \implies [B \rightarrow a_1 \ldots a_{3k} \ddagger b_1 \ldots b_n]$$
$$[B \rightarrow 111 a_1 \ldots a_{3k} \ddagger b_1 \ldots b_n] \implies [B \rightarrow a_1 \ldots a_{3k} \ddagger b_1 \ldots b_n]$$

(e) Finally, one metarule, to erase the assignment string and generate the accepting production:

$$[B \rightarrow \ddagger a_1 \ldots a_n] \implies [S \rightarrow \#]$$

As we mentioned above, clauses are erased iff they are true, and this metarule can only apply if all 3-CNF clauses in F have been erased. Thus, this metarule is used iff all the clauses are true, and the formula is satisfiable.

G contains the production $[S \rightarrow \#]$ iff F is satisfiable, so $\# \in L(G)$ iff F is satisfiable.

The result of applying any metarule (aside from those described in the first construction) is to change a basic rule so that the metarule cannot apply to it again. Note that the time required for the reduction is essentially the number of symbols needed to write the grammar down. The reduction can be performed in $O(m^3 \log m)$ time because the longest metarule is of length $O(m \log m)$, and there are $O(m^2)$ metarules. □

As promised earlier, I now prove that all possible truth assignments can be generated by the first metarule schema above, subject to the restriction that a metarule may only operate once in the derivation of a given rule. This is equivalent to proving that we can generate all binary numbers from 0 to $2^n - 1$ inclusive, using only $n + 1$ binary numbers and the metarules in 5.

Lemma 1 The $\frac{1}{2}n(n-1)$ metarules described by the schema 5, which can exchange any two bit positions i and j in a binary number of length n,

$$\begin{aligned} &\forall i, j, \quad 1 \leq i < j \leq n, \\ &[a_1 \ldots a_i \ldots a_j \ldots a_n] \implies [a_1 \ldots a_{i-1} a_j a_{i+1} \ldots a_{j-1} a_i a_{j+1} \ldots a_n] \end{aligned} \tag{5}$$

can generate all binary numbers from 0 to $2^n - 1$ using only the $n + 1$ binary numbers $1^i 0^{n-i}$, where i ranges from 0 to n (inclusive), even though no metarule may apply twice in the derivation of any given binary number.

Proof. Let $\alpha(k)$ be a binary number with k $1's$ in its binary representation, $0 \leq \alpha < 2^n$, and let $\beta(k) = 1^k 0^{n-k}$ be the k^{th} binary number. Then the following algorithm, expressed in a generic programming language, derives $\alpha(k)$ from $\beta(k)$ using the metarules:

PROCEDURE = DERIVE(α, β)

1: for $i = 1$ to $\lceil \frac{n}{2} \rceil$
2: if $\alpha_i \neq \beta_i$ then do
3: for $j = n$ to 0 step -1
4: if $\alpha_j \neq \beta_j$ then goto 6
5: next j
6: Let β be the result of applying the metarule
$[a_1 \ldots a_i \ldots a_j \ldots a_n] \Longrightarrow$
$[a_1 \ldots a_{i-1} a_j a_{i+1} \ldots a_{j-1} a_i a_{j+1} \ldots a_n]$
to β, switching bit positions i and j in the number β
7: next i

No metarule can be applied twice because i and j are different every time a metarule is applied (in line 6). In any derivation, at most $\lceil \frac{n}{2} \rceil$ metarules are applied (see line 1; in fact, exactly *MIN*$(k, n-k)$ metarules are applied in any derivation). □

Example derivation. Let $\alpha(5) = 0010011110$, a binary number with 5 $1's$ in its binary representation. Also let $\beta(5) = 1111100000$, the 5^{th} binary number. Then the following table illustrates how the algorithm of lemma 1 derives $\alpha(5)$ from $\beta(5)$.

β	metarule used	next β
$\underline{1}1111000\underline{0}0$	$[a_1a_2a_3a_4a_5a_6a_7a_8a_9a_{10}] \Longrightarrow [a_9a_2a_3a_4a_5a_6a_7a_8a_1a_{10}]$	0111100010
$0\underline{1}111000\underline{1}0$	$[a_1a_2a_3a_4a_5a_6a_7a_8a_9a_{10}] \Longrightarrow [a_1a_8a_3a_4a_5a_6a_7a_2a_9a_{10}]$	0011100110
$001\underline{1}10\underline{0}110$	$[a_1a_2a_3a_4a_5a_6a_7a_8a_9a_{10}] \Longrightarrow [a_1a_2a_3a_7a_5a_6a_4a_8a_9a_{10}]$	0010101110
$0010\underline{10}1110$	$[a_1a_2a_3a_4a_5a_6a_7a_8a_9a_{10}] \Longrightarrow [a_1a_2a_3a_4a_6a_5a_7a_8a_9a_{10}]$	0010011110

2.4 Example reduction from a 3SAT instance

This section uses an example reduction to provide a concrete illustration of the preceeding proof. Suppose the input 3SAT instance F is:

$$F = (a \vee \overline{b} \vee c) \wedge (\overline{a} \vee b \vee \overline{c})$$

Then the string of formula literals w is $a\overline{b}c\overline{a}b\overline{c}$, and the symbols q_1, q_2, and q_3 refer to the formula variables a, b, and c, respectively.

The reduction algorithm described above constructs the following GPSG in polynomial time in the length of F. Note that the metarule application process, and not the reduction algorithm, generates all possible truth assignments to F's variables.

1. 4 basic rules, and

$$[A \rightarrow 000000 \ddagger 000] \quad [A \rightarrow 000000 \ddagger 100]$$
$$[A \rightarrow 000000 \ddagger 110] \quad [A \rightarrow 000000 \ddagger 111]$$

2. 25 metarules

   (a) 3 metarules to generate assignments

$$[A \rightarrow 000000 \ddagger a_1 a_2 a_3] \Longrightarrow [A \rightarrow 000000 \ddagger a_2 a_1 a_3]$$
$$[A \rightarrow 000000 \ddagger a_1 a_2 a_3] \Longrightarrow [A \rightarrow 000000 \ddagger a_3 a_2 a_1]$$
$$[A \rightarrow 000000 \ddagger a_1 a_2 a_3] \Longrightarrow [A \rightarrow 000000 \ddagger a_1 a_3 a_2]$$

   (b) one metarule to freeze assignment generation

$$[A \rightarrow 000000 \ddagger a_1 a_2 a_3] \Longrightarrow [B \rightarrow 000000 \ddagger a_1 a_2 a_3]$$

   (c) 6 metarules to instantiate formula literals

   - 3 metarules for unnegated variables

$$[B \rightarrow 0 a_2 a_3 a_4 a_5 a_6 \ddagger 1 b_2 b_3] \Longrightarrow [B \rightarrow 1 a_2 a_3 a_4 a_5 a_6 \ddagger 1 b_2 b_3]$$
$$[B \rightarrow a_1 a_2 0 a_4 a_5 a_6 \ddagger b_1 b_2 1] \Longrightarrow [B \rightarrow a_1 a_2 1 a_4 a_5 a_6 \ddagger b_1 b_2 1]$$
$$[B \rightarrow a_1 a_2 a_3 a_4 0 a_6 \ddagger b_1 1 b_3] \Longrightarrow [B \rightarrow a_1 a_2 a_3 a_4 1 a_6 \ddagger b_1 1 b_3]$$

   - 3 metarules for negated variables

$$[B \rightarrow a_1 0 a_3 a_4 a_5 a_6 \ddagger b_1 0 b_3] \Longrightarrow [B \rightarrow a_1 1 a_3 a_4 a_5 a_6 \ddagger b_1 0 b_3]$$
$$[B \rightarrow a_1 a_2 a_3 0 a_5 a_6 \ddagger 0 b_2 b_3] \Longrightarrow [B \rightarrow a_1 a_2 a_3 1 a_5 a_6 \ddagger 0 b_2 b_3]$$
$$[B \rightarrow a_1 a_2 a_3 a_4 a_5 0 \ddagger b_1 b_2 0] \Longrightarrow [B \rightarrow a_1 a_2 a_3 a_4 a_5 1 \ddagger b_1 b_2 0]$$

(d) 14 metarules to verify guessed assignments, and

$$
\begin{array}{lcl}
[B \to 001 a_1 a_2 a_3 \ddagger b_1 b_2 b_3] & \Longrightarrow & [B \to a_1 a_2 a_3 \ddagger b_1 b_2 b_3] \\
[B \to 010 a_1 a_2 a_3 \ddagger b_1 b_2 b_3] & \Longrightarrow & [B \to a_1 a_2 a_3 \ddagger b_1 b_2 b_3] \\
[B \to 100 a_1 a_2 a_3 \ddagger b_1 b_2 b_3] & \Longrightarrow & [B \to a_1 a_2 a_3 \ddagger b_1 b_2 b_3] \\
[B \to 011 a_1 a_2 a_3 \ddagger b_1 b_2 b_3] & \Longrightarrow & [B \to a_1 a_2 a_3 \ddagger b_1 b_2 b_3] \\
[B \to 101 a_1 a_2 a_3 \ddagger b_1 b_2 b_3] & \Longrightarrow & [B \to a_1 a_2 a_3 \ddagger b_1 b_2 b_3] \\
[B \to 110 a_1 a_2 a_3 \ddagger b_1 b_2 b_3] & \Longrightarrow & [B \to a_1 a_2 a_3 \ddagger b_1 b_2 b_3] \\
[B \to 111 a_1 a_2 a_3 \ddagger b_1 b_2 b_3] & \Longrightarrow & [B \to a_1 a_2 a_3 \ddagger b_1 b_2 b_3]
\end{array}
$$

$$
\begin{array}{lcl}
[B \to 001 \ddagger b_1 b_2 b_3] & \Longrightarrow & [B \to \ddagger b_1 b_2 b_3] \\
[B \to 010 \ddagger b_1 b_2 b_3] & \Longrightarrow & [B \to \ddagger b_1 b_2 b_3] \\
[B \to 100 \ddagger b_1 b_2 b_3] & \Longrightarrow & [B \to \ddagger b_1 b_2 b_3] \\
[B \to 011 \ddagger b_1 b_2 b_3] & \Longrightarrow & [B \to \ddagger b_1 b_2 b_3] \\
[B \to 101 \ddagger b_1 b_2 b_3] & \Longrightarrow & [B \to \ddagger b_1 b_2 b_3] \\
[B \to 110 \ddagger b_1 b_2 b_3] & \Longrightarrow & [B \to \ddagger b_1 b_2 b_3] \\
[B \to 111 \ddagger b_1 b_2 b_3] & \Longrightarrow & [B \to \ddagger b_1 b_2 b_3]
\end{array}
$$

(e) one metarule to create the accepting production

$$[B \to \ddagger a_1 a_2 a_3] \Longrightarrow [S \to \#]$$

If we apply the metarules to the basic rules constructed above, a proper subset of the resultant context-free rules is:

$[A \to 000000 \ddagger 010]$ $[A \to 000000 \ddagger 001]$ $[A \to 000000 \ddagger 101]$ $[A \to 000000 \ddagger 011]$

$[B \to 000000 \ddagger 000]$ $[B \to 000000 \ddagger 001]$ $[B \to 000000 \ddagger 010]$ $[B \to 000000 \ddagger 011]$
$[B \to 000000 \ddagger 100]$ $[B \to 000000 \ddagger 101]$ $[B \to 000000 \ddagger 110]$ $[B \to 000000 \ddagger 111]$

$[B \to 010101 \ddagger 000]$ $[B \to 011100 \ddagger 001]$ $[B \to 000111 \ddagger 010]$ $[B \to 001110 \ddagger 011]$
$[B \to 110001 \ddagger 100]$ $[B \to 111000 \ddagger 101]$ $[B \to 100011 \ddagger 110]$ $[B \to 101010 \ddagger 111]$

$[B \to 101 \ddagger 000]$ $[B \to 100 \ddagger 001]$ $[B \to 110 \ddagger 011]$ $[B \to 001 \ddagger 100]$
$[B \to 000 \ddagger 101]$ $[B \to 011 \ddagger 110]$ $[B \to 010 \ddagger 111]$

$[B \to \ddagger 000]$ $[B \to \ddagger 001]$ $[B \to \ddagger 011]$ $[B \to \ddagger 100]$
$[B \to \ddagger 110]$ $[B \to \ddagger 111]$

$[S \to \#]$

The formula F is satisfiable, because the production $[S \to \#]$ is generated by metarule application.

3 The EP Paradox Resolved

At first glance, a proof that GPSG-Recognition is NP-hard appears to contradict Gazdar's efficient parsability argument noted above. GPSGs only generate CFLs and CFLs can be recognized in polynomial time ($O(n^3)$ for a sentence of length n). Therefore it would seem that GPSGs can also be recognized in polynomial time, simply by converting the target GPSG into a weakly equivalent context-free grammar (CFG) and recognizing using that CFG.

This argument is misleading because it ignores both the effect converting the GPSG into a CFG has on grammar size, and the effect grammar size has on recognition speed. The crux of the matter is that even a highly constrained GPSG grammar can result in an exponentially larger derived context-free rule set. Informally, each metarule application can more than double the size of the GPSG grammar G. Since there are $O(|G|)$ metarules, the resulting derived grammar can be of size $O(|G| \cdot 2^{|G|})$, that is, exponentially larger than the GPSG.[3] Standard context-free parsers like the Earley algorithm actually run in time $O(|G|^2 \cdot n^3)$ where $|G|$ is the CFG size and n the sentence input length, so the hypothetical GPSG grammar G will be recognized in time

$$O((|G| \cdot 2^{|G|})^2 \cdot n^3)$$

Even if the GPSG grammar is held constant, the exponential increase in derived grammar size will result in an astronomical constant multiplicative factor, which will dominate the performance of the Earley algorithm for all expected inputs (that is, those of a million words or less), every time we use the derived grammar. Thus, in the worst case, if a GPSG with 10 symbols recognized a given sentence in .001 second, a grammar with 50 symbols would recognize the same sentence in 35.7 years, and a grammar with 100 symbols could take at least 10^{15} centuries.[4] (Gazdar's (1981) toy grammar

[3]This mathematical analysis is vindicated in practice. Phillips and Thompson (1985:252) observe that in their parser based on the GPSGs of Gazdar (1982), "To expand the [GPSG] grammar completely . . . would be ridiculously wasteful of space and time. (The toy grammar of English we use with GPSGP [their parser], of 29 phrase-structure rules and four metarules, which expands to 85 rules, is equivalent to several tens of millions of context-free rules.)" Similarly, Shieber (1983:137) notes that typical post-Gazdar(1982) GPSG systems contain "literally trillions" of derived rules. Ristad (1986b:83) estimates that the GKPS grammar for English corresponds to at least 10^{33} context-free productions.

[4]Evans (1985:237) experiences the real-world intractability of GPSG-Recognition first hand in his GPSG-based parser, and proposes to manage it by eliminating lexical am-

contained many hundreds of symbols, so a grammar size of 100 is somewhat small.)

The resolution of the EP paradox may also be understood as a special case of the central distinction between *problem complexity* and *algorithm complexity*. As I mentioned earlier, the complexity of a *problem* is its *inherent complexity*, the computational cost of solving a problem, no matter which existing or undiscovered algorithm is used. Conversely, the complexity of an *algorithm* is the cost of a specific algorithm or procedure for solving a problem. Thus, the fact that GPSGs can succinctly encode some CFGs indicates straightforward use of standard CFG recognition *algorithms* will fail to be efficient for GPSGs, because a GPSG is weakly equivalent to a very large CFG, and CFG size affects recognition time; yet that fact in no way bears on the complexity of the GPSG recognition *problem*. The complexity result, on the other hand, firmly establishes that no known or yet to be invented algorithm for GPSG-Recognition will be efficient, unless $\mathcal{P} = \mathcal{NP}$.

Although known grammar conversion procedures increase both the grammar size and recognition time for the GPSG, the preceding discussion does not in principle preclude the possibility of "compiling" the GPSG into a "fast" grammar.[5] If the compiled grammar is truly fast and assigns the same structural descriptions as the uncompiled GPSG, and it is possible to compile the GPSG in practice, then the complexity of the universal RP would not accurately reflect the real cost of parsing. But until such a suggestion is forthcoming, I assume that it does not exist.

4 Restricting Metarule Application

Since the central problem is that GPSG metarules are capable of deriving any finitely large set of rules, including exponentially large ones, we must further constrain metarule application if we wish to solve the GPSG-

biguity and by keeping both grammar and input string size as small as possible: "The attempts to overcome the time and space problems have only been partially successful . . . . The only remedies seem to be, keep phrases as short as possible (for example, do not try to test large noun phrases inside complex sentences if it can be avoided — use proper nouns instead), make sure no words are duplicated in the lexicon, keep the number of ID rules currently loaded down where possible . . . ."

[5] Barton (1985) shows how grammar expansion increases both the space *and* time costs of recognition, when compared to the cost of using the grammar directly.

Recognition problem in polynomial time and thereby obtain an efficient parsability result.

A list of restrictions necessary to remove GPSG-Recognition from the class of NP-hard problems is:[6]

- strictly bounded "chaining" — only a constant number of metarules, fixed in advance for all GPSG grammars, can operate in the derivation of a given context-free rule.
- each metarule may derive a rule set only polynomially bigger than its input rule set.
- a metarule may only use "abbreviatory variables."

5 Defining the Recognition Problem

Following complexity theory practice, I use the *universal* recognition problem — given a grammar G and an input string x, is $x \in L(G)$? — to formally analyze GPSG's efficient parsability (EP) claims. Alternately, the recognition problem (RP) for a class of grammars may be defined as the *fixed language* RP (FLRP): given an input string x, is $x \in L$ for some fixed language L? For the FLRP, it does not matter which grammar is chosen to generate L — typically, the fastest grammar is picked.

It seems reasonably clear that the universal RP is of greater linguistic and engineering interest than the FLRP. The grammars licensed by linguistic

[6] In order to guarantee that these three restrictions are sufficient, GPSG must be completely and exactly formally specified, in a manner which ensures that proliferation of categories will not make the recognition problem intractable. Another aspect of current GPSG formulations which make them NP-hard — and probably intractable — is the immediate dominance/linear precedence (ID/LP) formalism. See Barton (1985) for a proof. Note that the linguistically untenable restriction of prohibiting metarule variables of any kind is probably sufficient, when coupled with restrictions on ID/LP, to guarantee polynomial time recognition. Such a restriction would mean that a metarule, which may only "match" one basic rule, can only derive exactly one rule. The size of the derived context-free rule set would be the size of the basic rule set plus the number of metarules. This restriction is linguistically unmotivated because it fails to capture linguistically important generalizations. For example, any metarule applying to singular and plural sentences would have to be replicated at least twice: once to handle the singular case, and once to handle the plural case.

theory assign structural descriptions to utterances, which are interpreted semantically, translated into other human languages, and so on. The universal RP, unlike the FLRP, determines membership with respect to a grammar, and therefore more accurately models the parsing problem, which must use a grammar to assign structural descriptions.

The universal RP also bears most directly on issues of natural language acquisition. The language learner evidently possesses a mechanism for selecting grammars from the class of learnable natural language grammars $\mathcal{L}_G$ on the basis of linguistic inputs. The more fundamental question for linguistic theory, then, is "what is the recognition complexity of the class $\mathcal{L}_G$?". If this problem should prove computationally intractable, then the (potential) tractability of the problem for each language generated by a G in the class is only a partial answer to the linguistic questions raised.

Finally, complexity considerations favor the universal RP. The goal of a complexity analysis is to characterize the amount of computational resources (e.g. time, space) needed to solve the problem *in terms of all computationally relevant inputs.* We know that both input string length and grammar size and structure affect the complexity of the RP. Hence, excluding either input from complexity consideration in order to argue that the RP for a family of grammars is tractable would not advance our scientific understanding.[7]

Linguistics and computer science are primarily interested in the universal RP because both disciplines are concerned with the formal power of a *family* of grammars. Barton, Berwick, and Ristad (1986, forthcoming) elaborates and extends these arguments.

[7]This "consider all relevant inputs" methodology is universally assumed in the formal language and computational complexity literature. For example, Hopcroft and Ullman (1979:139,346) define the context-free grammar RP as "Given a CFG $G = (V, T, P, S)$ and a string x in T^*, is x in $L(G)$?", and the context-sensitive language RP as "Given a CSG G and a string w, is w in $L(G)$?" Garey and Johnson (1979) is a standard reference work in the field of computational complexity. All 10 automata and language recognition problems covered in the book (pp. 265-271) are universal, i.e. of the form "Given an instance of a machine/grammar and an input, does the machine/grammar accept its input?" The complexity of these RPs is *always* calculated in terms of grammar and input size.

6 The Complexity of Succinctness

Section 3 resolved the EP paradox with an argument that superficially linked representational succinctness with computational complexity: exponential grammar expansion retards CFG recognition times when using standard algorithms. Therefore, it may be tempting to blame intractability on expressive economy. However, there is no causal relationship between succinctness and intractability — simply because the two notions are mathematically distinct.

Complexity results characterize the amount of resources needed to solve instances of a problem, while succinctness results measure the space reduction gained by one representation over another, equivalent, representation. There is no casual connection between computational complexity and representational succinctness, either in practice or principle. In practice, converting one grammar into a more succinct one can either increase or decrease the recognition cost. For example, converting an instance of context-free recognition (known to be efficient) into an instance of context-sensitive recognition (thought to be intractable) can significantly speed the RP if the conversion decreases the size of the CFG logarithmically or better. Even more strangely, increasing ambiguity in a CFG can speed recognition time if the grammar size is reduced sufficiently and slow it down otherwise — unambiguous CFGs can be recognized in linear time $O(|G|^2 \cdot n)$, while ambiguous ones can require cubic time $O(|G|^2 \cdot n^3)$. Berwick and Weinberg (1982) discuss these issues in greater detail.

In principle, tractable problems may involve succinct representations. For example, the iterating coordination schema (ICS) of GPSG is an unbeatably succinct encoding of an infinite set of context-free rules; from a computational complexity viewpoint, parsing with the ICS is utterly trivial using a slightly modified Earley algorithm.[8] Tractable problems may also include verbose representations: consider a random finite language, which may be recognized in essentially constant time on a typical computer, yet whose elements must be individually listed. Similarly, intractable problems can involve either succinct or nonsuccinct representations. As is well known, the Turing machine for an arbitrary recursively enumerable set may be ar-

[8] A more extreme example of the unrelatedness of succinctness and complexity is the absolute succinctness with which Σ^*, the dense language of all strings over the alphabet Σ, may be represented — whether by a regular expression, CFG, or even Turing machine — yet members of Σ^* may be recognized in constant time (i.e. always accept).

bitrarily big or small.

The complexity result shows that GPSGs are not merely succinct encodings of some context-free *grammars*; they are inherently complex grammars for some context-free *languages*.

7 Conclusion

A central goal of this paper has been to define the framework within which efficient parsability claims are best evaluated. Gazdar (1981) claims to offer the beginnings of an explanation for efficient parsability, yet the universal recognition problem for the GPSGs of Gazdar (1981) is provably NP-hard. Inasmuch as his argument overlooks a computational problem that is likely to be intractable, any support for GPSG on this basis is extremely weak. Metarules, even when severely constrained, are one source of GPSG's complexity.

The moral of this result is that as far as we know casual appeal to general mathematical results is not likely to rescue efficient parsability results. Specific constraints on the particular representations postulated by linguistic theory are needed to explain efficient linguistic processing. This does not imply that GPSG theory is without merit: on the contrary, I have merely shown that its particular efficient parsability thesis cannot be maintained. Generalized phrase structure grammar, lexical functional grammar, and transformational grammar are all probably intractable in an abstract mathematical sense, and each theory must search elsewhere for an explanation of efficient natural language parsing, if one is to be given at all.[9]

8 References

Barton, G. Edward (1985). "On the Complexity of ID/LP Parsing," *Computational Linguistics,* 11:205–218.

Barton, G. Edward, Robert Berwick, and Eric Ristad (1986, forthcoming). *Computational Complexity and Natural Language.* Cambridge, MA: MIT Press.

[9]Berwick and Weinberg (1984) discusses the complexity of LFG.

Berwick, Robert and Amy Weinberg (1982). "Parsing Efficiency, Computational Complexity, and the Evaluation of Grammatical Theories." *Linguistic Inquiry* 13:165–191.

Berwick, Robert and Amy Weinberg (1984). *The Grammatical Basis of Linguistic Performance*. Cambridge, MA: MIT Press.

Earley, Jay (1970). "An Efficient Context-Free Parsing Algorithm," *Communications of the ACM* 13:94–102.

Evans, Roger (1985). "ProGram — a development tool for GPSG grammars," *Linguistics* 23(2):213–243.

Garey, Michael, and Johnson, David (1979). *Computers and Intractability.* San Francisco: W.H. Freeman and Co.

Gazdar, Gerald (1981) "Unbounded Dependencies and Coordinate Structure," *Linguistic Inquiry* 12:155–184.

Gazdar, Gerald (1982). "Phrase Structure Grammar," in P. Jacobson and G. Pullum (eds.), *The Nature of Syntactic Representation.* Dordrecht: Reidel.

Gazdar, Gerald, Ewan Klein, Geoffrey K. Pullum, and Ivan A. Sag (1985). *Generalized Phrase Structure Grammar.* Oxford, England: Basil Blackwell.

Hopcroft, John E., and Ullman, Jeffrey D. (1979) *Introduction to Automata Theory, Languages, and Computation.* Reading, MA: Addison-Wesley.

Joshi, Aravind (1985) "Tree Adjoining Grammars." In *Natural Language Parsing*, D. Dowty, L. Karttunen, and A. Zwicky, eds. Cambridge: Cambridge University Press.

Lewis, Harry and Christos Papadimitriou (1978) "The Efficiency of Algorithms," *Scientific American* 238:96–109.

Peterson, I. (1985) Exceptions to the rule. *Science News* 128:314-315.

Phillips, John D. and Henry S. Thompson (1985). "GPSGP — a parser for generalized phrase structure grammars," *Linguistics* 23:245–261.

Ristad, Eric S. (1986a) "Computational Complexity of Current GPSG Theory," *Proceedings of the 24th Annual Meeting of the ACL*, pp.30–39.

Ristad, Eric S. (1986b) "Complexity of Linguistic Models: A computational analysis and reconstruction of generalized phrase structure grammar." S.M. Thesis, Department of Electrical Engineering and Computer Science, M.I.T.

Shieber, Stuart M. (1983) "Direct Parsing of ID/LP Grammars," *Linguistics and Philosophy* 7:135–154.

Shieber, Stuart M., Susan U. Stucky, Hans Uszkoreit, and Jane J. Robinson (1983). "Formal Constraints on Metarules," *Proceedings of the 21st Annual Meeting of the ACL*, pp.22–27.